A cryptographic key management architecture for dynamic 6LowPan networks

Ruben Smeets[a], Kris Aerts[a], Nele Mentens[a], Dave Singelee[c], An Braeken[b], Laurent Segers[b], Abdellah Touhafi[d], Kris Steenhaut[d], Niccolo De Caro[d]

[a]KU Leuven@KHLim
Firstname.Lastname@Kuleuven.be

[b]Vrije Universiteit Brussel(VUB)
FirstnameLastname@vub.ac.be

[c]KU Leuven, ESAT/COSIC & iMinds
Firstname.Lastname@esat.Kuleuven.be

[d]Vrije Universiteit Brussel, ETRO
FirstnameLastname@etro.vub.ac.be

Abstract

Wireless sensor networks are becoming an important facilitator for the Internet of Things. These embedded devices can harvest different types of information such as temperature, pressure and humidity, which offer important data for making decisions regarding various applications such as healthcare, logistics and smart homes. Different sensors working together act as a local sensor network. With the advent of the new 6LowPan standard the sensor nodes can even participate in Internet communications, opening up even more possibilities. The downside is that these networks are more prone to intrusion by unwanted parties. Furthermore implementing security is not straightforward due to the constrained nature of the sensor nodes, although different solutions have been proposed. One of the remaining and most challenging issues is the key management problem. In this paper, we propose a symmetric key management scheme for wireless sensor networks that uses tamper-proof hardware for key generation and distribution. The scheme requires no deployment knowledge before enrolling and makes use of a trusted central entity for key negotiation to provide end-to-end security. Our implementation and evaluation were performed on the tiny Zolertia Z1 hardware platform, running Contiki-OS. The performance and security evaluation show that our scheme requires a limited amount of storage and provides a good network resilience against node capture.

Keywords: 6LowPan, Wireless Sensor Network Security, Symmetric Key Management, Link-Layer Security, End-to-End Security

MSC: 68-00, 68-01, 94-02, 94A60, 94A62

1. Introduction

The recently released 6LowPan standard allows for a tight integration between the Internet and wireless sensor networks (WSNs) [2]. Sensor nodes using 6LowPan can directly communicate with IPv6 hosts which enables the use of standard servers for sensor data processing. This considerably increases the flexibility of wireless sensors and reduces the need for complex gateways. On the other hand, granting Internet connectivity introduces additional security implications. Aside from the known attacks against WSNs, these sensor networks now have to deal with security problems of traditional IP networks [1]. Furthermore, the constrained nature of the wireless devices limits the use of complex security mechanisms. Nevertheless security is generally required in WSNs. For instance, in a home automation application, where adversaries can use the temperature measurements to deduct the absence of inhabitants. In such a setting, encryption would be desirable to ensure data confidentiality [1].

Nowadays, wireless sensor nodes with network connectivity generally run an operating system (OS) to provide multiple application support. One of these OSs is Contiki [3], which is an open source, C-based, multitasking OS for memory-efficient networked embedded systems and WSNs. It uses the μIPv6-stack for 6LowPan communications and is growing in popularity. Contiki, however, does not provide built-in support for security key management.

In this paper we present our open source Contiki implementation of a symmetric key management scheme suited for a variety of sensor nodes. The scheme offers network-wide integrity [2], confidentiality and end-to-end security as well as semantic security [3]. It relies on a basic infrastructure for key distribution and makes use of tamper-proof hardware for key generation. The implementation and evaluation are performed on the Zolertia Z1 hardware platform, running Contiki-OS [4]. In our design we tried to take full advantage of the available hardware security primitives from the CC2420 transceiver by Chipcon [5]. Being able to use a hardware AES128 co-processor [6] for security operations, releases more time and resources for a robust implementation of the key management scheme. This is reflected in an improved resilience against denial-of-service (DoS) attacks.

The outline of the paper is organized as follows. In the next section, we review the related work in WSNs. In Sect. 3 we present the proposed key management architecture. The implementation of the scheme within Contiki is described in Sect. 4. In Sect.5 we evaluate and analyze the performance of our implementation. After the evaluation, we conclude the paper in Sect. 6.

[1] Confidentiality: prevents an attacker from reading what is being transmitted over the communication channel.

[2] Integrity: guarantees that the message has not been modified during the transmission.

[3] Semantic security: knowledge of the ciphertext (and length) of some unknown message does not reveal any additional information on the message that can be feasibly extracted

2. Related work

Consulting the literature on implementations of symmetric key management architectures shows that SPINS is one of the first security architectures designed for WSNs [7]. It uses two secure building blocks called SNEP and μTesla to provide data confidentiality, authentication, integrity and freshness. We used this work as a starting point for our design, due to the low resource requirements and good scaling factors. Unfortunately, SNEP was never fully specified.

Yüksel et al. present the Zigbee symmetric key management scheme for low power WSNs [8]. The scheme provides network security and end-to-end security, by using three different types of keys: a master key, a network key and a link key. They specify the symmetric-key key establishment (SKKE) protocol for key negotiation and the mutual entity authentication (MEA) protocol for authentication. We used architectural design choices of this work in our design.

In 2009, ContikiSec [9] was presented as a secure network layer for WSNs, integrated in the Contiki-OS. They provide three security modes: confidentiality, authentication and integrity. In addition, they offer a thorough measuring methodology that we used during our evaluation.

The contribution of our work is providing an open source implementation of a symmetric key management scheme suited for constraint WSNs running the Contiki-OS.

3. Proposed key management architecture

In this section, we present our symmetric key management scheme specifically designed for low-power IP-enabled WSNs. After analyzing the possible attacks against WSNs [1], we chose for a distributive scheme that uses parts of the SPINS protocol [7] and the Zigbee 2007 security scheme [8]. We start by defining the general topology of the network in Sect. 3.1. Followed by the various phases of the scheme in Sect. 3.2 and, finally, the security properties in Sect. 3.3.

3.1. The key management topology

The key management scheme infrastructure can be divided into four major parts. The first part consists out of a Tablet and portable PUF [4]. They are used during the initial phases of the scheme for key generation and distribution. The second part is the trusted central entity (TCE), which serves as the main dissemination component for cryptographic material in the network. The third part is the 6LowPan edge-router (ER) that acts as the seamless interconnection between the central entity and the WSN. Finally, we have the individual sensor nodes. The above mentioned parts are illustrated in Figure 1. All of them play a particular role

[4] PUF: a Physical Unclonable Function is a physical entity that is embodied in a physical structure and is easy to evaluate but hard to predict.

throughout the different phases of the scheme, by making use of three types of keys: the network key (Kn), sensor key (Ks) and session key (Kss).

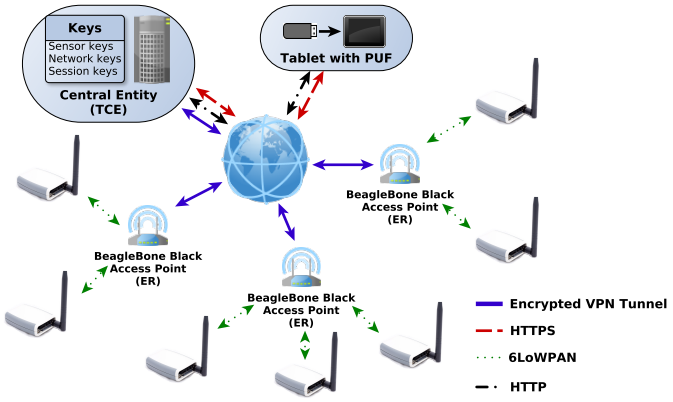

Figure 1: Topology of the key management scheme

3.2. Key management scheme phases

Our scheme consists of three phases: the bootstrapping, the discovery and the pairing phase.

Bootstrapping phase

The bootstrapping phase describes how the network is initialized and installed. We start by setting-up the database of the TCE. The PUF in combination with the tablet, generates a list of cryptographic strong keys for use inside the entire network. The list of keys is then transferred to the TCE through a secure connection and stored in the database. Once the central entity has enough security material it is possible to connect the 6LowPan ERs. Each ER sends a network-key request over an encrypted tunnel completing the initialization of the basic key management infrastructure.

After the basic infrastructure is initialized, we can start adding sensor nodes to the 6LowPan networks. A new sensor node has no knowledge of the designated network and therefore has no associated cryptographic key material for that network. In order to add a new node we need to bootstrap the right key material. This is done by using a wired interface that connects the node to the TCE as shown in Figure 2. The node sends a hello-packet over a serial SLIP connection to the installer laptop, which on its turn forwards the packet over the encrypted tunnel to the central entity. The information from the node is stored in the database and awaits approval from the user. Once the user has authorized the node through a tablet interface, the TCE will transfer the selected key material back over the same connection. Each node receives a unique Ks and a Kn that is shared by the assigned 6LowPan network. The Ks is used for Kss establishment and will be

discussed later on.

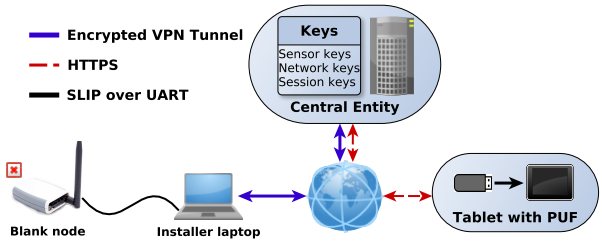

Figure 2: Wired bootstrapping of new sensor nodes

Discovery phase
During the discovery phase the sensor nodes try to establish their routes by means of a routing protocol. The shared Kn is used to secure and authenticate the communications to and from each node. The entities and adversaries that have no knowledge of the Kn are unable to participate in network communications. This is a basic security feature of the IEEE 802.15.4 standard [10] and increases the robustness of the network.

Pairing phase
The pairing phase describes how two entities can establish a Kss by using a variant of the SNEP protocol [7] based on the Needham–Schroeder symmetric protocol [11]. Suppose that a node 'x' wants to establish a Kss with node 'y' through a trusted third party, which in our case is the trusted central entity (TCE). The TCE plays a role as key distribution center and shares an individual Ks with each node in the network. Node x will initiate the communication by sending a request message to node y. Node y receives the request and replies with a challenge. This challenge, in combination with additional data from node x, is transferred to the TCE by node x. The TCE will perform the authentication and select the Kss. Afterwards it will send the Kss back to nodes x and y, encrypted with their individual Ks. In order to verify if the key transfer was successful, node x demands a verification message from node y. If the message is correct, the protocol is completed successfully. Once the pairing phase is finished, the nodes can start using their shared Kss for secure end-to-end communications.

3.3. Security properties

The key-management scheme properties are listed in Table 1. We give an overview of the advantages and security trade-offs that exist in our scheme.

The security mechanisms of our scheme operate at two different levels of the μIP-stack: the link layer and application layer. In the former, the shared Kn is used by the AES core in CBC-MAC mode to provide message integrity. In the latter, the Kss achieves end-to-end security by using the AES-core in CCM mode, providing data confidentiality, authentication and semantic security [12].

Advantages	Security trade-offs
- Optimal use of hardware security primitives. - Nodes have limited responsibilities. - Storage of few keys in each node. - Compromising a node forms no threat for end-to-end security. - No pre-shared keys required between nodes. - Excluding compromised nodes is trivial. - Good scaling factors due to limited storage of keys.	- Adding nodes forms threat due to unencrypted serial SLIP interface. - Manual malicious node detection. - Adding nodes is slow. - Compromised Kn breaks link layer security. - Central entity is single point of trust. - Central entity contains all cryptographic material. - Relies on basic infrastructure for key management.

Table 1: Overview of the security properties of the proposed scheme

4. Implementation in Contiki

The key-management scheme is implemented as a separate process within the multi-threaded environment of Contiki. It operates completely autonomous from other layers within the stack allowing transparency for the application. Figure 3 displays the current μIP-stack involvement of the scheme. The core functionality is situated in a layer on top of the transport layer, while the wired bootstrapping mechanism is done separately. We start by discussing the bootstrapping protocol, followed by a detailed description of our key-management implementation.

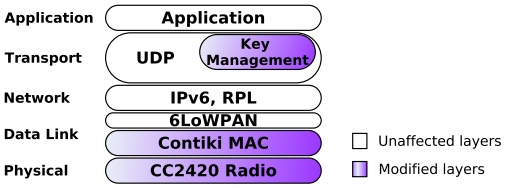

Figure 3: μIP-stack overview

The wired bootstrapping protocol uses the existing SLIP component within Contiki to save memory resources. The component has been modified to solely fit the requirements of the protocol. The program starts by checking the content of the external flash memory for cryptographic key material. We used the external memory to store the Kn, Ks and nonce data as well as the TCE address to prevent the loss of bootstrap material in case of a power failure or reset. If there are no keys detected, the SLIP component is initialized and the hello-packet, containing the node information, is transmitted once. As mentioned in the previous section, the packet is then forwarded and awaits approval from the user. After permission, the TCE formulates a reply message that consist of the Ks, Kn and TCE-address and the message is sent back. The running SLIP process handles the reply message

and stores the security data in the node's external flash memory. The final step is performing a software reset to load the new security material and finalize the bootstrap protocol. This time the SLIP component is not initialized since there is valid security material. The advantages of disabling the SLIP component after bootstrapping are: we deny adversaries to use the serial interface after deployment and we free-up processing power.

The key-management layer is implemented as a separate process that contains two event-handlers running in parallel. The first is a polling handler using the event-timer library of Contiki. The second is a udp-event handler that gets triggered when receiving a packet on the key management udp-connection. Both events make use of security material that is arranged in slots described in Table 2. The slots contain the following information about the sessions: nonce data, Kss, remote ipv6 address, time of last activity and a status byte. The first two slots are reserved for communications with TCE. The third slot, Slot 2, is reserved for temporary key exchange data and is only used during key exchange. The remaining session slots are used for storage of end-to-end security data and can be expanded at compile time. Notice that increasing the slot size has a negative impact on memory.

Slot 0	Slot 1	Slot 2	Slot 3	Slot 4	Slot 5
TCE Ks	TCE Kss	RESV	Session 0	Session 1	Session 2

Table 2: Key management security material slots

The main key management program involves a state machine that gets manipulated by the events, shown in Figure 4. During the IDLE state, the polling event periodically checks the status of the slots for changes regarding nonce updates and key requests. If a session in one of the slots is expired the state will be changed to key request, invoking the key exchange protocol. The udp-events can also change the state to key request if a valid request message is received. The expired/requested session data is copied to the RESV slot and the protocol is initialized. Since there is only one RESV slot available, there can only be one key exchange at a time.

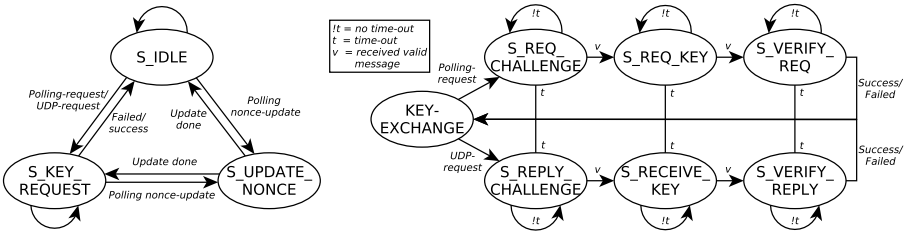

Figure 4: Main state machine (left) and Key state machine (right)

The used exchange protocol, described in the previous section, is also implemented as a state machine that is illustrated in Figure 4. Depending on the action,

expired or requested, a different path is chosen by the key state machine. Each key-state remains locked until a new valid message is received or a time-out occurs. This prevents unwanted behavior in case of replay packets or broken communication links. During the lock, protocol messages are retransmitted until the maximum retransmission count is exceeded or the lock is released. After the time-out, the protocol is aborted and the session request is marked as failed. The failed sessions are put in a queue to allow other requests. Eventually, after repeated failure of a queued request, the failed session is assigned a time-out that increases with the amount of failures. If, on the other hand, the protocol is concluded successfully, the RESV data is copied to the requesting slot or a free session slot. By using the lock on states and the time-out on session failures, we increase the resilience against DoS attacks. Consider, for example, continuous interference of adversaries on the channel preventing the protocol from finishing successfully. Our implementation prevents the exhaustion of battery power due to the increased time-out on each failure.

The encryption and decryption functions also reside in the key management layer. The application uses these functions to perform the end-to-end security operations and request new sessions. In contrast to predetermining a list of communication sessions, we decided to implement an on-demand architecture allowing the application to dynamically choose its sessions at runtime. The encryption function is partially responsible for handling this functionality. Each time the application wants to send an encrypted message to a new destination, the function searches the slots for a free spot or makes room for one by removing the least active session. The new session is marked as expired and in turn the polling mechanism captures the session request. Another way of overwriting a session slot is by successfully concluding a remote key request. If the requesting node is unknown and the slots are full, the least active device is overwritten with RESV data. The advantage of this method is the flexibility without the need for large memory requirements and the increased resilience against node capture. Consider, for example, a configuration where we have three session slots available. Even if there are ten compromised nodes in the network, all willing to establish a Kss with us, the application is still able to communicate with the desired entities by overwriting the compromised slots. The disadvantage, however, is the time and energy cost that is bound to the continues reestablishment of a new Kss due to session slot overflow.

The link-layer security and CC2420 AES-CCM drivers are implemented in the lower layers of the μIP-stack. We only had to change small settings in the radio driver to enable link-layer security. The AES-drivers ensure that all the block cipher modes are performed in hardware by the AES co-processor of the CC2420.

5. Evaluation and results

Resource management is one of the most important aspects of WSNs. Therefore, we measured the impact of encryption/decryption speed, energy consumption as well

as memory usage of our implementation, by using the same measuring methodology as described in ContikiSec [9]. We start by listing the encryption/decryption speeds together with the energy consumption, shown in Table 3.

	Speed (ms)	energy (μJ)
AES-CCM driver Enc.	1.30 $\pm$0.01	115
AES-CCM driver Decr.	1.44 $\pm$0.01	124
End-to-end AES-CCM Enc.	1.71 $\pm$0.01	120
End-to-end AES-CCM Decr.	1.55 $\pm$0.01	126

Table 3: Speed and energy results for the security operations

The results of the security operations are measured for a message containing 50 bytes of associated data, 69 bytes of encrypted payload and 8 bytes for the message integrity code. We took the average of 100 speed measurements to formulate our results. Looking at the datasheet of the CC2420 transceiver, we see that the AES-CCM encryption itself takes 222 μs. This translates in 1.08 ms overhead caused by the driver for copying data and setting registers of the transceiver. If we add key-management overhead, we lose 1.49 ms for encryption and 1.33 ms for decryption.

The memory requirements of the scheme and security mechanisms are described in Table 4. We used the 20-bit addressing capabilities of the MSP430F2617 [13] to enable the full 92kB of program memory.

	Contiki (no 20-bit)	Contiki (with 20-bit)	AES-CCM driver	Link-layer security	Full impl.
ROM (B)	48927	58099	986	2202	7282
RAM (B)	5778	6762	/	446	557

Table 4: Memory requirements of the implementation

The utilization of 20-bit addressing is translated in an increase of almost 19% in memory. This results in a total size 58 kB for the Contiki-OS without security. The AES-CCM driver accounts for 986 bytes of the full key management implementation. The Link-layer security and wired bootstrapping takes 2.2 kB of the program memory. Finally, the full program size of our symmetric key management implementation is 7282 bytes of ROM and 557 bytes of RAM. In total, our scheme costs 8% of the available memory space of our platform.

6. Conclusion

We presented a robust implementation of our symmetric key management scheme, which provides link-layer security and end-to-end security for WSNs. The scheme has been integrated in the μIP-stack of the Contiki operating system. Our design

tries to achieve low energy consumption combined with a high resilience against DoS attacks. Our evaluation was performed on the Zolertia Z1 platform, a general purpose development platform created by Zolertia. In the future we will try to extend the scheme with key-updates and examine the possibility of secure wireless bootstrapping for the sensor nodes.

Acknowledgements. This work is funded by the IWT-TETRA project 120105: 6LoWPAN - Towards zero-configuration for wireless building automation.

References

[1] Çayirci, E., Rong, C., Security in Wireless Ad Hoc and Sensor Networks, *Published by John Wiley & Sons Ltd.*, (2009), 131–283.

[2] Shelby, Z., Bormann, C., 6LoWPAN: The Wireless Embedded Internet, *Published by John Wiley & Sons Ltd.*, (2009).

[3] Dunkels A., Gronvall B., Voigt T., Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors, *In: Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (IEEE Computer Society, Washington)*, (2004).

[4] Zolertia, Zolertia Z1 datasheet, *http://zolertia.sourceforge.net/wiki/images/e/e8/Z1_RevC_Datasheet.pdf*

[5] Chipcon, CC2420 datasheet, *http://www.ti.com/lit/ds/symlink/cc2420.pdf*

[6] Daemen J., Rijmen V., The Design of Rijndael: AES - The Advanced Encryption Standard, *Springer-Verlag*, (2002)

[7] A. Perrig, R. Szewczyk, J.D. Tygar, V. Wen, D.E. Culler, Spins: Security protocols for sensor networks, *Wirel. Netw. 8(5)*, (2002), 521–534.

[8] E. Yüksel, H.R. Nielson, and F. Nielson, ZigBee-2007 Security Essentials, *In: Proceedings of the 13th Nordic Workshop on Secure IT Systems (NordSec)*, (2008), 65–82.

[9] Casado, L., Tsigas, P., ContikiSec: A Secure Network Layer for Wireless Sensor Networks under the Contiki Operating System, *In: Proceedings of the 14th Nordic Conference on Secure IT Systems (NordSec)*, (2009).

[10] IEEE 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks(LR-WPANs), *3 Park Avenue, New York, USA: IEEE*, (2003)

[11] Needham RM, Schroeder MD, Using encryption for authentication in large networks of computers, *Communications of the ACM 21 (12)*, (1978), 993–999

[12] National Institute Of Standards and Technology Recommendation for Block Cipher Modes of Operation *(SP800-38A)*, March (2012).

[13] Texas Instruments, MSP430F2617 datasheet, *http://www.ti.com/lit/ds/symlink/msp430f2617.pdf*